

SDEV1201

Database Fundamentals

LESSON 15

Let's review

CREATE TABLE **minimum syntax**

11

```
CREATE TABLE TableName (  
    Column1 DATATYPE  
    , Column2 DATATYPE  
    , Column3 DATATYPE  
    ...  
)
```

where *Column#* represents the name of the attribute/column and *DATATYPE* is *INT*, *DATETIME*, etc.

Constraints are used to:

21

1. Define the **PK**
 - This is defined by the **primary key** constraint
2. Define **relationships**
 - This is defined by the **foreign key** constraint
3. Define **default values**
 - Defined by the **default** constraint
4. Define a **domain of valid values**
 - Defined by the **check** constraint
5. Ensure that **all values in a column are unique**

Primary Key Constraint

When using a normalized database design, all tables must have a primary key constraint. Any column that acts as a primary key must be defined as a not null column. The DBMS will ensure that all rows in the table have a unique value for primary key columns.

When a primary key constraint is defined, PostgreSQL automatically creates a unique index for the column(s) acting as the primary key.

Syntax

Column constraint syntax:

[Constraint constraint_name] Primary Key

Table constraint syntax:

[Constraint constraint_name]

Primary Key (col_name [, col_name2 [, . . . col_name32]])

FK Constraints & referential integrity

The foreign key constraint defines referential integrity between two tables. Operations that affect the tables and/or the data stored within the tables must comply with referential integrity. This affects:

1. Dropping tables.
2. Creating tables.
3. Inserting/Updating/Deleting rows in tables.

The bottom line for referential integrity is that a child row/table cannot exist without its associated parent row/table. Therefore:

1. You must Drop a child table BEFORE you Drop the associated parent table.
2. You must Create a parent table BEFORE you Create the associated child table.
3. The value of a column acting as a foreign key must be:
 - A value that exists as a primary key in the associated parent table, or
 - NULL

Foreign Key Constraint Syntax

Column constraint syntax:

[Constraint constraint_name]
References ref_table (ref_column)

Table constraint syntax:

[Constraint constraint_name]
Foreign Key (column_name [, . . . column_name32])
References ref_table (ref_column [, . . . ref_column32])

- column_name identifies the column(s) acting as the foreign key
- ref_table identifies the parent table
- ref_column identifies the primary key in the associated parent table

Indexes

An index is an object stored in the db.

It's associated with 1 or more columns, in a specific table, that act as **the key for the index**.

This helps the DBMS look up (retrieve) the data quicker.

Functionally an index object is similar to an index at the back of a book.

Syntax

```
Create [Unique] Index [If Not Exists] index_name  
    On table_name ( column_name [, ...n] [ Asc | Desc] );
```

Column name represents the key for the index. One or more columns may act as the key for the index. All indexes must have a unique name. In SDEV1201 we normally use the prefix **IX** when naming an index.

Default Constraint

The default constraint lets you define a value that is assigned to a column when the user does not specifically supply a value when a new row is added to the table (using the Insert statement). A default constraint can be defined on any column except a column with a serial property.

The value of the default can be supplied via a constant (5.90) or a function (i.e. Current_Date). The default constraint name should use a prefix of DF (indicating this is a default constraint).

Syntax:

[Constraint constraint_name]
Default {constant | function}

Unique Constraint

The unique constraint makes sure that the values in one or more columns are not repeated. When applied to a single column, this ensures all entries in that column are different. When applied to multiple columns, this requires that the combination of values across those columns is unique, so no two rows can have exactly the same values.

A unique constraint does not consider null values to be equal.

Unique Constraint

Adding a unique constraint will automatically create a unique index on the column or group of columns used in the constraint.

Column Constraint Syntax

`[Constraint constraint_name] Unique`

Table Constraint Syntax

`[Constraint constraint_name]`

`Unique (col_name [, col_name2 [ , . . . col_name32] ] )`

Check Constraint

The **CHECK** constraint enables you to specify what values are acceptable (i.e. define a DOMAIN).

CHECK constraints should be given a meaningful name and consist of an expression that evaluates to TRUE or FALSE.

Syntax:

```
CONSTRAINT CK_ConstraintName CHECK (expression)
```

Whenever a new row of data is added to the table, or an existing column has its value changed the new data will be evaluated using the expression in the check constraint. If the expression evaluates to true the DBMS will add the new data to the table. If the expression evaluates to false an error message is issued and the new data is rejected.

ALTER TABLE

- The **ALTER TABLE** statement is used to:
- Add a column (with or without constraints)
- Drop an existing column (assuming you do not break any referential integrity such as PK - > FK)
- Change the datatype of an existing column
- Add a default value to an existing column
- Drop a default from an existing column
- Add/Drop a Not Null constraint to an existing column
- Drop an existing constraint
- Add a table constraint
- Rename a column, or column constraint, or table

Data Manipulation Language

Generally, DML commands fall into one of three primary categories:

- INSERT adds fresh data to a table.
- UPDATE Change the data that is already in a table.
- DELETE takes a record out of a table.

The INSERT Statement

The INSERT statement adds 1+ rows to a table.

```
INSERT [INTO] TableName (column1, column2, ...)  
VALUES (value1, value2, ... )
```

Values can be hardcoded values, expressions, DEFAULT, NULL, or even a subquery.

Alternatively, we can INSERT data from another table: we use the results of a SELECT query as the values to INSERT.

```
INSERT [INTO] TableName (column1, column2, ...)  
SELECT ...
```


Update

The Update statement is used to update existing data in a table. It can be used to update one or more columns. Data can be updated by providing new values for columns or by using subqueries to provide the new data.

Update

Simplified Syntax:

```
UPDATE table_name
SET column = expression [ , column = expression . . . ]
[FROM table_name1 [, table_name2 ...] ]
[WHERE expression];
```

table_name - identifies the table to be updated

SET - identifies the column(s) to be updated and the expression which provides the new value for the column. The expression can be a subquery.

FROM clause - optional

- identifies additional table(s) referenced in the where clause that lets you update rows in a table based on data in one or more other tables. If more than one table is included, tables are comma-delimited

WHERE clause - optional. Identifies what records are to be updated. if the FROM clause is utilized, table join condition(s) are coded here

Delete

The Delete statement removes rows from a table. It does not act on individual columns as you can only delete a row and not a column.

Delete

Simplified Syntax:

```
DELETE [From] table_name  
[USING clause]  
[WHERE clause];
```

[From] - optional

- makes the statement more English-like

table_name - required

- identifies the table you want to delete rows from

USING clause - optional

- identifies additional table(s) referenced in the where clause that lets you delete rows in a table based on data in one or more other tables. If more than one table is included, tables are comma-delimited

WHERE clause- optional

- identifies which rows to delete from the table
- if not coded **ALL** rows are deleted if the Using clause is utilized, table join condition(s) are coded here

Today's Objective

By the end of this section we will cover:

- ❑ Declare and use variables
- ❑ Write IF ELSE statements
- ❑ **Quiz 3**

Documentation can be found in Brightspace in the “Week 10 & 11 – Stored Procedures” section under Materials. “SQL Flow Control And Variables (PostgreSQL Version)”.

Assignment 3 (TH : Create Table & DML Statements) is available on Brightspace and is **due Friday, October 31, 2025 at 5:00 PM.**

Anonymous Code Blocks

Anonymous Code Blocks

Why use DO?

Run multi-statement logic without creating a stored function/procedure.

Perfect for data fixes, quick loops, conditional DDL, and ad-hoc scripting.

Run multi-statement logic without creating a stored function/procedure.

Perfect for data fixes, quick loops, conditional DDL, and ad-hoc scripting.

Anonymous Code Blocks

--\$\$ create the block for the code that you will execute

DO \$\$

Declare --A section to declare variables

my_message TEXT:= 'Hello World!'; --Declares a variable called my_message and assigns 'Hello World to it'

my_mood TEXT:= 'groovy';

Begin--The executable statements go here

Raise Notice '% Hope you have a % day!', my_message,my_mood;

--Raises a message to pgamdins messages tab or whatever output console you are working in. It is mainly for testing and debugging.

--% is a place holder

--You can also format a raise Notice as

Raise Notice using message = my_message ||' Hope you have a ' || my_mood || 'day!';

End \$\$;

Anonymous Code Blocks

You can control the flow of code as well! (IF)

If condition Then

Begin

-- SQL statements

End

ElsIf another_condition Then

Begin

-- SQL statements

End

Else

Begin

-- SQL statements

End

End If;

Variables

SQL Flow Control And Variables

You may use variables and flow control statements in a stored procedure or a batch. Besides being in a stored procedure, you can execute a group of SQL statements simply by highlighting the statements you want to execute. They could contain multiple statements with flow control and variables. They do not have to be inside a stored procedure. However, they need to be in a stored procedure if you wish to call them from outside SQL (such as a client application).

SQL Flow Control And Variables

PostgreSQL supports the use of variables and flow control statements within stored procedures, functions, or anonymous code blocks (using Do blocks).

You can execute a group of SQL statements (including flow control and variables) without creating a stored procedure by using a Do block. However, if you want to call your code from outside SQL (for example, from a client application), then you should place it inside a stored procedure or function. \$\$ (double quoting) is a PostgreSQL substitute for single quotes to avoid quoting issues inside the Begin...End block. This means that you can use single quotes around strings without any issues.

Variables

Variables

Variables are used to hold and manipulate values in memory. They must be declared before they are used, allowing PostgreSQL to allocate storage efficiently.

Syntax

Declare

```
variablename datatype;  
variablename datatype;  
variablename datatype;
```

All variables must be declared at the beginning of the block, before any executable statements.

Variables

Assigning Values to Variables

Variables can be assigned literal values or values from queries.

Assigning a Literal Value

```
variable_name := literal_value;
```

Example

Example:

Do \$\$

Declare

v_Student_Count integer;

Begin

v_Student_Count:= 100;

Raise Notice 'Number of students: %', v_Student_Count;

End \$\$;

Raise Notice in PostgreSQL sends a message to the client or message window. It is similar to Print in SQL Server but should mainly be used for testing or debugging, not for returning data to the user.

Variables

Assigning a Value from a Query

```
SELECT column_name  
INTO    variable_name  
FROM    table_name  
WHERE   condition;
```


Example

Do \$\$

Declare

V_Total_Pay Decimal(8,2);

Begin

Select Sum(Amount)

Into V_Total_Pay

From Payment;

Raise Notice 'Total Payments: %', V_Total_Pay;

End \$\$;

Flow Control

Flow Control

You can control the execution of SQL statements using flow control structures such as If, Elself (optional), and Else (optional). You can have as many Elself statements as you want, but only one Else statement is allowed. You can use Begin and End for readability, but it is not required in If statements.

Flow Control

Syntax:

If **condition** Then

 Begin

 -- SQL statements

 End

Elsif **another_condition** Then

 Begin

 -- SQL statements

 End

Else

 Begin

 -- SQL statements

 End

End If;

Example

Do \$\$

Declare

 v_Avg_Mark decimal(5,2) := 75;

Begin

 If v_Avg_Mark >= 80 Then

 Raise Notice 'Excellent!';

 ElsIf v_Avg_Mark >= 50 Then

 Raise Notice 'Pass';

 Else

 Raise Notice 'Fail';

 End If;

End \$\$;

Flow Control

Using If Exists / If Not Exists

A common use of conditional logic is to check if a record exists in a table before performing an action. The “If Exists” clause returns True if the subquery returns at least one record, and False otherwise. Conversely, using “If Not Exists” returns False if the subquery returns at least one record, and True otherwise.

Syntax:

If Exists (Select 1 From **table_name** Where **condition**) Then

-- SQL statements

End If;

Example

```
Do $$  
Begin  
  If Exists (Select 1  
    From Student  
    Where StudentID = 123) Then  
    Raise Notice 'Student already exists.';  
Else  
  Raise Notice 'Student does not exist.';  
End If;  
End $$;
```

Change **123** to **199899200** to see if you get a different message.

Flow Control

Anonymous Code Blocks (Do)

Each Do block or stored procedure executes as a single unit.

You can separate logical sections of a script simply by executing one block at a time.

Hypothetical Example (Client table does not exist in IQSchool):

Do \$\$

Begin

 Alter Table Client

 Add Column Active Char(1);

End \$\$;

Do \$\$

Begin

 Alter Table Client

 Add Constraint CK_Client_Active_TF Check (Active In ('T', 'F'));

End \$\$;

In this example, each Do block executes separately. This ensures the column is added before the constraint is created.

Flow Control

Notes

- Do blocks are executed immediately and are not stored in the database.
- Use Create Procedure or Create Function if you need to store reusable code in the database.